Research Article

The Concept Drift Problem in Android Malware Detection and Its Solution

**Donghui Hu,¹ Zhongjin Ma,¹ Xiaotian Zhang,¹ Peipei Li,¹
Dengpan Ye,² and Baohong Ling³**

¹College of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China

²College of Computer Science, Wuhan University, Wuhan 430072, China

³Anhui Broadcasting Movie and Television College, Hefei 230011, China

Correspondence should be addressed to Donghui Hu; hudh@hfut.edu.cn

Received 10 June 2017; Accepted 7 August 2017; Published 18 September 2017

Academic Editor: Zheng Yan

Copyright © 2017 Donghui Hu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Currently, the Android platform is the most popular mobile platform in the world and holds a dominant share in the mobile device market. With the popularization of the Android platform, large numbers of Android malware programs have begun to emerge on the Internet, and the sophistication of these programs is developing rapidly. While many studies have already investigated Android malware detection through machine learning and have achieved good results, most of these are based on static data sources and fail to consider the concept drift problem resulting from the rapid growth in the number of Android malware programs and normal Android applications, as well as rapid technological advancement in the Android environment. To address this problem, this work proposes a solution based on an ensemble classifier. This ensemble classifier is based on a streaming data-based Naive Bayes classifier. Android malware has identifiable feature utilization tendencies. On this basis, feature selection algorithm is introduced into the ensemble classifier, and a sliding window is maintained inside the ensemble classifier. Based on the performance of the subclassifiers inside the sliding window, the ensemble classifier makes dynamic adjustments to address the concept drift problem in Android malware detection. The experimental results from the proposed method demonstrate that it can effectively address the concept drift problem in Android malware detection in a streaming data environment.

1. Introduction

With the continuous development of mobile technology, smartphones have become indispensable tools for daily life and work. Due to its openness, the Android platform has attracted large numbers of developers and users and become the most popular mobile platform in the world. As the number of Android users continues to grow, the Android platform market has caught the eye of a considerable number of malware developers. According to a report by G DATA (a cybersecurity company), the number of new Android malware samples reached 560,671 in the second quarter of 2015, a 27% increase over the first quarter. On average, 6,100 new Android virus samples emerged every day (equivalent to a new virus every 14 seconds) in the second quarter of 2015, 1,200 more than in the first quarter

(<https://www.gdata-china.com/news/article/article/mmre-porth2>). Thus, Android malware has become one of the main threats to Android users.

The rapid growth in Android malware has already garnered attention from researchers in China and elsewhere. To date, many studies on Android malware detection through machine learning have been conducted. However, all of these studies are based on static data sources and fail to consider the concept drift problem that may exist as a result of the rapid development of Android malware and normal Android applications, as well as the technological advancement in the Android environment. A frequently encountered challenge when using machine learning to solve practical problems is the fact that a problem may rely on hidden contexts that are not explicitly provided by its features. Changes in the hidden contexts will have spreading effects on the

problem. This phenomenon is referred to as concept drift [1]. For example, in the weather forecast, it is difficult to judge whether the weather is sunny or cloudy when we only know the temperature and do not know the season. The temperature of 25 degrees Celsius is likely to be cloudy in summer, and it is likely to be sunny in winter. Concept drifts are mainly reflected by changes in data distribution.

In a previous study [2], we noticed that the concept drift problem also exists in Android malware detection. According to an early study by Zhou and Jiang [3], compared with normal applications, Android malware has a tendency to use specific features to achieve its malicious intents, such as sending fee-based text messages, encoding and decoding, and dynamic code loading. However, in our statistical analysis of the features used by normal Android applications and by Android malware, we found an interesting phenomenon: some features that are often used by malware, such as dynamic loading, also frequently appear in normal application software for the purpose of code protection. We believe that this phenomenon is not a special case. With the improvement of smartphone hardware and the increasing demands of users, the functions of normal Android applications will continuously expand. At the same time, Android malware developers will continuously adjust the technologies used in malware based on existing detection methods. Data distribution in the entire sample space will undergo continuous changes over time. If a classifier obtained by training on an old data source continues to be used, a large number of errors will be generated when this classifier is used to classify samples that are newly generated from data sources whose underlying distribution has changed. In addition, the number of Android malware programs and normal Android applications grows very fast. The characteristics of Android data samples are consistent with those of streaming data fast, continuously, infinitely, and variably. Therefore, a new solution should be formulated based on the characteristics of streaming data and Android malware detection.

In this work, a Naive Bayes Classifier Based on Streaming Data (NBCS) is proposed. The NBCS is formed by adding detection technology to a Naive Bayes classifier based on the characteristics of streaming data. This ensures that the training samples that the classifier obtains from streaming data are sufficient in number to represent the entire sample space. In addition, because Android malware exhibits certain feature utilization tendencies, a feature selection algorithm is also incorporated into the NBCS to enhance its performance. Based on the NBCS, a model for Ensemble Learning Based on Random NBCS (ENBCS) is proposed. The ENBCS model allocates a random feature subset to each subclassifier. Each subclassifier selects features from the feature subset with which it is provided. The ENBCS model also uses a window-based subclassifier elimination strategy, by which the accuracy of each subclassifier within the window is monitored, and subclassifiers with relatively low accuracy are eliminated. Afterward, subclassifiers are retrained using the data within the window to address the concept drift phenomenon.

The remainder of the paper is structured as follows: Section 2 introduces work related to Android malware detection, as well as work related to concept drift detection.

Section 3 focuses on the relevant details of the NBCS and ENBCS algorithms. Section 4 presents experiments that demonstrate the performance of the NBCS and ENBCS algorithms. Section 5 presents the conclusions of this work and provides recommendations for future work.

2. Relevant Work

The key to addressing concept drift lies in detecting the occurrence of concept drift and providing measures with which it can be addressed. Hulten et al. [4] proposed a concept-adapting very fast decision tree learner (CVFDT) based on the very fast decision tree algorithm [5]. The CVFDT dynamically updates the statistical value required for each node in the Hoeffding tree (HT) by using a sliding window. When concept drift occurs, the CVFDT will select a new feature for each node based on the updated statistical value and establish new subtrees. Rutkowski et al. [6] noted the deficiencies of the CVFDT algorithm in the HT and proposed a McDiarmid tree (McDT) algorithm by replacing the Hoeffding bound in the HT with McDiarmid's bound. Rutkowski et al. [7] also addressed the relatively long computation time of the McDT algorithm using Gaussian approximation. In the NBCS, a Naive Bayes classifier, which can be quickly trained, is used as the basic classifier. A feature selection algorithm is also introduced into the NBCS to improve its performance. Gama et al. [8] treated the error rate of a classifier as a binomial distribution and proposed a drift detection method (DDM). In addition, they set a warning level and a drift level based on different confidence levels. Furthermore, they treated the data between the warning and drift levels as a window with which to retrain classifiers. Based on Gama et al.'s DDM, Baena-García et al. [9] proposed an early DDM (EDDM) to address the deficiencies of Gama et al.'s DDM in detecting sudden concept drifts. Bifet and Gavalda [10] proposed an adaptive windowing (ADWIN) algorithm. The ADWIN algorithm divides a sliding window into two sufficiently large subwindows and detects concept drifts by comparing the data in the two subwindows. The ENBCS algorithm determines whether a concept drift occurs by comparing the accuracy of the ensemble classifier and the subclassifiers within the window.

Because concept drifts are often related to time, many studies have addressed the concept drift problem by sample weighting. A time-based sample weighting method has been previously proposed, in which the newest sample is assigned the greatest weight [11, 12]. Zhang et al. [13] classified concept drifts into strict and loose concept drifts. For loose concept drifts, they assigned weights to samples using the kernel mean matching method. For strict concept drifts, they proposed an optimal weights adjustment (OWA) method for assigning weights to samples. Instead of assigning weights to samples, the ENBCS algorithm solves the concept drift problem by directly ignoring the samples outside the window. Another approach in addition to sample weighting is available for solving the concept drift problem: subclassifier weighting through ensemble learning. A streaming ensemble algorithm (SEA) [14] processes a data flow in blocks. Upon acquiring a new data block, an SEA uses this data block to train a

new classifier, compares the new classifier with the previously trained classifier, and retains the classifier with better performance. Some researchers [15, 16] have used a weighted voting method to classify each sample in ensemble learning models. The weight of each subclassifier will be updated based on its accuracy. Bifet et al. [17] enumerated all the feature subsets of size k , used these feature subsets in parallel to train the HT classifier, input the results obtained from the HT classifier into a perceptron, and trained it using the gradient descent method. Bifet et al. also used the ADWIN algorithm to detect concept drifts and reset the learning rate of the gradient descent method upon the occurrence of a concept drift. The ENBCS algorithm allocates a random feature subset to each subclassifier, and each subclassifier selects features from the feature set with which it is provided. The feature sets used by the subclassifiers vary in size. Classifiers with relatively low accuracy are eliminated directly, after which classifiers are retrained.

3. Problem Description and Solution

In this work, a two-tuple, (X, C) , is used to represent a data sample, where $X = \{x_1, x_2, \dots, x_n\}$ represents the n -dimensional characteristics of the sample and $C \in \{C_1, C_2, \dots, C_n\}$ represents the type tag of the sample.

3.1. Problem Description. When using Bayesian theory to classify sample $X = \{x_1, x_2, \dots, x_n\}$, C_i is selected as the classification tag and satisfies the following condition:

$$\begin{aligned} C_i &= \arg \max_{i \in \{1, \dots, m\}} P(C_i | X) \\ &= \arg \max_{i \in \{1, \dots, m\}} \frac{P(x_1, x_2, \dots, x_n, C_i)}{P(X)}. \end{aligned} \quad (1)$$

However, the topological structure of the Bayesian network is required to calculate $P(x_1, x_2, \dots, C_i)$. Learning the Bayesian network from the data is very difficult, and the time complexity of computation increases exponentially. To simplify the calculation process, in a Naive Bayes classifier, features are assumed to be independent of one another under the given classification tag (C) condition. Thus, (1) can be transformed to

$$\begin{aligned} C_i &= \arg \max_{i \in \{1, \dots, m\}} P(C_i | X) \\ &= \arg \max_{i \in \{1, \dots, m\}} \prod_{j=1}^n P(x_j | C_i) P(C_i). \end{aligned} \quad (2)$$

For data source S , when its distribution remains stable within a consecutive t number of time periods (i.e., $S_1 = S_2 = \dots = S_t$), $P_1(x_j | C_i)$ and $P_1(C_i) = P_t(C_i)$. The prior probabilities obtained by learning on S_1 , $P_1(x_j | C_i)$, and $P_1(C_i)$ can be used to predict the data sample generated on S_t (X_t). However, in the real environment where Android malware detection is performed, the number of Android malware programs and normal Android applications grows extremely fast. In addition, technologies used in Android malware programs

and normal Android applications are continuously evolving. Thus, there is a time point, t' , at which $S_1 = S_2 = \dots = S_t \neq S_{t'}$, $P_1(x_j | C_i) \neq P_{t'}(x_j | C_i)$, and $P_1(C_i) \neq P_{t'}(C_i)$. As a result, the classifier obtained by training on S_1 cannot effectively detect data samples generated on $S_{t'}$. In this work, this phenomenon is referred to as the concept drift phenomenon associated with Android malware detection.

3.2. Solution

3.2.1. NBCS. As demonstrated in (2), it is necessary to first learn each conditional probability, $P(x_i | C_j)$, to use the NBCS to classify samples. For a static data sample, the conditional probability can be easily obtained. However, in a streaming data model, data samples are acquired sequentially, and the entire data space is infinite. Therefore, when processing streaming data, one of the primary tasks is to determine whether sufficient data samples have been acquired. To complete this task, the NBCS is proposed. It is assumed that, in the overall sample space, $P(x_i | C_j) = \mu_{ij}$, where μ_{ij} is unknown. $P'_t(x_i | C_j), P'_{t+1}(x_i | C_j), \dots, P'_{t+k}(x_i | C_j)$ are $k+1$ observed values of $P(x_i | C_j)$. According to the central limit theorem, the distribution corresponding to an observed value, $P'_t(x_i | C_j)$, should follow a normal distribution with an expected value of μ_{ij} , that is, $P'_t(x_i | C_j) \sim N(\mu_{ij}, \sigma_2)$. Because μ_{ij} is unknown, a transformation is performed. Let $\Delta P_o(x_i | C_j) = P'_a(x_i | C_j) - P'_b(x_i | C_j), \forall a, b \in \{t, \dots, t+k\}$ and $a \neq b$. Because $P'_a(x_i | C_j)$ and $P'_b(x_i | C_j)$ are independent and identically distributed random variables, the distribution of the difference between $P'_a(x_i | C_j)$ and $P'_b(x_i | C_j)$ should follow a normal distribution with an expected value of $\mu_{ij} = 0$. Based on hypothesis testing, null and alternative hypotheses are made as follows: $H_0 : \mu = 0$ (null hypothesis) and $H_1 : \mu \neq 0$ (alternative hypothesis). A statistic is selected:

$$T = \frac{\Delta \bar{P} - 0}{S/\sqrt{k}} \sim t(k-1), \quad (3)$$

where

$$\begin{aligned} \Delta \bar{P} &= \frac{1}{k} \sum_{o=1}^k \Delta P_o(x_i | C_j), \\ S &= \sqrt{\frac{1}{k-1} \sum_{o=1}^k (\Delta P_o(x_i | C_j) - \Delta \bar{P})^2}. \end{aligned} \quad (4)$$

For a given confidence level, α , because $P\{|T| \geq T_{\alpha/2}\} = \alpha$, the rejection region can be obtained:

$$I_c = \{t | |t| \geq t_{\alpha/2}\}. \quad (5)$$

For the feature of each dimension, x_i , the NBCS maintains a window with a size of $k+1$ to store the observed values of the most recent $k+1$ conditional probabilities, $P'_t(x_i | C_j)$. Every time a new sample enters the classifier, the numerical values in each feature window are dynamically updated. If H_0 holds, then for the data samples that have

```

Input: A sequence streaming data  $L = \{X_1, X_2, \dots, X_n, \dots\}$ 
 $F$ , feature set used in this classifier,  $|F| = m$ 
 $\alpha$ , the confidence level
window, the size of window
 $\delta$ , the threshold used in feature selection

(1) /* Init */
(2) count = 0
(3) for  $i = 1$  to  $m$  do
(4)    $\text{Flag}_i = 0$  /* the state of features, 0 indicates that
      the observation value  $P'(x_i | C_j)$  could not stand for
       $P(x_i | C_j)$  */
(5) end for
(6) while count < window AND  $\exists i, \text{Flag}_i = 0$  do
(7)   count ++
(8)   for  $F_i \in F$  do
(9)     if  $\text{Flag}_i == 0$  then
(10)      Calculate  $t$  using equation (3)
(11)      if  $|t| \geq t_{\alpha/2}$  then
(12)        Calculate  $\text{SU}(F_i, C)$  using equation (6)
(13)        if  $\text{SU}(F_i, C) > \delta$  then
(14)           $\text{Flag}_i = 1$  /* 1 indicates that this feature will
            be used in classification */
(15)        else
(16)           $\text{Flag}_i = 2$  /* 2 indicates that this feature does
            not pass feature selection */
(17)        end if
(18)      end if
(19)    end if
(20)  end for
(21) end while

```

ALGORITHM 1: Training stage of the NBCS algorithm.

been acquired, the average of the observed values of the conditional probabilities of the feature in question can be used to represent the conditional probability value of the entire data space, $P(x_i | C_j)$. In some special cases, the classifier acquires a large number of data samples, and some features cannot pass the null hypothesis. To prevent this, the NBCS sets a receiving window for training samples for each classifier. When the number of samples from which a classifier has learned exceeds the window size, the classifier will mark the features that fail to pass the null hypothesis and will not consider these features in subsequent feature selection and sample classification processes.

3.2.2. Feature Selection. Most Android malware programs are repackaged versions of normal software applications [3]. These repackaged applications will retain the normal functions of the original applications. Therefore, Android malware programs will retain the features of the original applications. In addition, to achieve its malicious intent, malware has a tendency to use certain features. Based on this characteristic of Android malware, irrelevant and redundant features can be eliminated through feature selection. Moreover, based on a Naive Bayes classifier, the NBCS assumes that features are independent of one another under the given type (C) condition. However, in practical applications, this

assumption rarely holds. Eliminating redundant features and weakening the correlation among features through feature selection can, to a certain extent, enhance the classification performance of the NBCS. In this work, the FCBC algorithm [18] is selected as the feature selection algorithm. In the FCBC algorithm, symmetrical uncertainty (SU) [19] is used as the standard for evaluating the correlation between any two features. Let X and Y be two random variables. Then, the SU between X and Y is as follows:

$$\text{SU}(X, Y) = 2 \left[\frac{I(X, Y)}{H(X) + H(Y)} \right], \quad (6)$$

where $H(X)$ represents the information entropy of X :

$$H(X) = - \sum_i P(x_i) \log_2(P(x_i)). \quad (7)$$

$I(X, Y)$ represents the information gain between X and Y :

$$I(X, Y) = H(X) - H(X | Y). \quad (8)$$

The value of SU falls in the range of $[0, 1]$. The higher the value of $\text{SU}(X, Y)$, the more significant the correlation between X and Y . In the FCBC algorithm, a threshold value, δ , needs to be specified. If $\text{SU}(X, Y) > \delta$, then X is correlated with Y ; otherwise X is not correlated with Y . Algorithm 1 shows the learning process of the NBCS classification model in pseudocode.

```

Input: A sequence streaming data  $L = \{l_1, l_2, \dots\}$ ,  $|l_i| = M$ 
 $N$ , the number of classifiers in the ensemble learning model
 $m$ , the number of features used in each classifier
 $\alpha$ , the confidence level
 $K$ , window size of the ensemble learning model
(1) while more data blocks  $l_t$  are available do
(2)   Getting the accuracy rate  $P_{en,t}$  of ensemble model on  $l_t$ 
(3)   for  $i = 1$  to  $N$  do
(4)     Get the accuracy rate  $P_{i,t}$  of classifier  $i$  on  $l_t$ 
(5)     Calculate  $T_i$  from (9)
(6)     if  $T_i < 0$  AND  $|T_i| \geq t_\alpha$  then
(7)       /* Classifier  $i$  should be abandoned */
(8)       Retrain classifier  $i$  with data  $\{l_t, l_{t-1}, \dots, l_{t-K+1}\}$ 
(9)     end if
(10)  end for
(11) end while

```

ALGORITHM 2: Subclassifier elimination strategy used in the ENBCS.

3.2.3. ENBCS Model. Ensemble learning is a machine learning method with the purpose of obtaining better learning results than those obtained using single learners by training a series of subclassification models and subsequently integrating each learning result with a certain rule [20–22]. Ensemble learning plays a role in conventional static machine learning. Many researchers who study concept drifts have used ensemble learning and achieved good experimental results [1, 14, 17].

In this work, an ENBCS model is proposed based on the NBCS. The ENBCS provides a randomly selected feature subset to each NBCS, and these feature subsets are the same size. Each subclassifier selects features from the feature subset with which it is provided. A maximum voting method is used in the ENBCS model to classify test samples. Because the features learned by a subclassifier model are a feature subset that is randomly extracted from the overall feature space, it is very likely that this feature is not sufficiently capable of expressing the class attributes. As a result, some subclassifiers have low accuracy. A subclassifier elimination method is used in the ENBCS model: the classification results of each subclassifier in the ensemble learning model are examined in real time, and subclassifiers with poor classification results are eliminated. Algorithm 2 shows this method in pseudocode.

When classifying data samples, the ENBCS model first divides the streaming data into data blocks of size M before processing them. In the ENBCS model, a window of size K is specified to store K data blocks ($l_{t-K+1}, l_{t-K+2}, \dots, l_t$) as well as the accuracy of the ENBCS for these data blocks ($P_{en,t-K+1}, P_{en,t-1}, \dots, P_{en,t}$). For any arbitrary classifier i , a window of size K is maintained correspondingly to store the accuracy for each data block within the window. Each time the ensemble classifier obtains a new data block, l_t , the accuracy of the ensemble classifier and that of each subclassifier are calculated for this data block, and, subsequently, hypothesis testing is performed to determine whether the accuracy of each subclassifier within the window is significantly lower

than that of the ensemble classifier at a given confidence level α . A statistic is selected:

$$T_i = \frac{\bar{P}_i - \bar{P}_{en}}{S/\sqrt{K}} \sim t(K-1), \quad (9)$$

where

$$\begin{aligned} \bar{P}_{en} &= \frac{1}{t} \sum_{i=1}^t P_{en,i}, \\ \bar{P}_i &= \frac{1}{K} \sum_{j=t-K+1}^t P_{i,j}, \\ S &= \sqrt{\frac{1}{K-1} \sum_{j=t-K+1}^t (P_{i,j} - \bar{P}_i)^2}. \end{aligned} \quad (10)$$

For the given confidence level α , if $T_i < 0$ and $|T_i| \geq t_\alpha$, then the accuracy of the subclassifier in question is considered to be significantly lower than that of the ensemble classifier, and this subclassifier should be eliminated and retrained using the data within the window. When solving the concept drift problem, some researchers [8] believe that, for a group of test data, the error rate of classification is a random variable that follows a binomial distribution. When there are sufficient samples, this binomial distribution can be approximated by a normal distribution. Under the condition that the distribution of samples remains stable, the error rate of a learning algorithm will gradually decrease as the number of samples increases. A significant increase in the error rate of a learning algorithm indicates a change in the data distribution. Based on this theory, the subclassifier elimination strategy used in the ENBCS is also applicable to solving the concept drift problem. When concept drift occurs, the accuracy of the subclassifiers within the window will decrease significantly. According to (9) and (10), when $T_i < 0$ and $|T_i| \geq t_\alpha$, subclassifiers will be eliminated, and new training data will be learned within the window.

4. Experimental Results and Analysis

4.1. Data Source. The data used in this experiment came from three sources. The first part of the data was obtained from the Drebin dataset. We randomly extracted 3,000 samples from the Drebin dataset (<https://www.sec.cs.tu-bs.de/~danarp/drebin/index.html>), of which 809 are malware samples and 2,191 are normal software samples. The third part of the data was collected independently and is composed of 3,000 samples, of which 808 are malware samples and 2,192 are normal software samples. The aforementioned two batches of samples are discontinuous in time. Between them is a mixed dataset, which is composed of 862 samples, of which 144 were obtained from the Drebin dataset and 718 were collected independently. In this experiment, the mixed samples in the middle are viewed as the time point at which concept drift occurs.

4.2. Feature Set. The feature set used in this work is determined based on previous research and has 405 dimensions. The feature set is mainly composed of three parts. The first part of the feature set is permissions for which Android applications apply. Permissions are a very important part of Android software. If an Android application needs to call a system service and read the private information of the user, the application needs to apply for the corresponding permission in the AndroidManifest.xml file. The Android operating system will read the permission application information when the Android application is being installed and ask the user whether the corresponding permission should be granted. If the Android application fails to obtain the needed permission, the corresponding code will not be run. Currently, this type of feature has 152 dimensions.

The second part of the feature set is actions used by Android applications. In Android software, actions are used to register and monitor various events in the system. When an event occurs, the Android component in which the event is registered will be triggered. Information about actions is also written in the AndroidManifest.xml file. This type of feature has 229 dimensions. The third part of the feature set consists of application programming interfaces (APIs) and some Linux commands used by Android applications. An Android application cannot complete specific functions by only applying for permissions; it also needs to use API calls provided by the Android system based on the needs of the program. An Android development framework provides thousands of API calls, of which most features are unrelated to Android malware detection. Previous studies [23, 24] are used as references for this part of the feature set. This type of feature has 24 dimensions.

4.3. Analysis of the Performance of the NBSC Algorithm. At the feature selection stage, a threshold value, δ , needs to be specified for the FCBC algorithm. If the value set for δ is too small, the classification model cannot effectively eliminate redundant and unrelated features. If δ is too large, the number of features learned by the NBSC algorithm will be too small, and thus no effective predictions will be

FIGURE 1: Accuracy of the NBSC algorithm under various threshold values.

FIGURE 2: Feature selection results obtained using the NBSC algorithm under various threshold values.

made. Figure 1 shows the accuracy of the NBSC algorithm when δ is set to various values (threshold: none signifies that no feature selection is performed). Figure 2 shows the number of features of each of the three types under various window sizes. These three types correspond to the marks in Algorithm 1. In Figure 2, flag: 0 signifies the features that failed to pass the null hypothesis; flag: 1 signifies the features

FIGURE 3: Comparison of the accuracy of the NBCS algorithm and the selected comparative algorithms.

used for classification prediction; and flag: 2 signifies the features that failed to pass the feature selection procedure. As demonstrated in Figure 2, when no feature selection is performed and δ is relatively small, there are a relatively large number of redundant and irrelevant features in the classifier. As a result, the classification performance of the NBCS algorithm is not ideal. As δ increases, the number of features used by the classifier (flag: 1) begins to decrease and the accuracy of the NBCS algorithm begins to improve. However, when δ exceeds a certain value (e.g., 0.4), while the features used by the classifier are highly correlated with one another, the total number of features is too small (as seen from the rightmost bar of the histogram), and the accuracy of the classifier decreases. In subsequent experiments, δ is set to 0.1.

In this work, the HT algorithm and the Hoeffding Option Tree (HOT) algorithm (an improved HT algorithm) [25] are used as comparative algorithms. These two classifier algorithms are realized in Massive Online Analysis (MOA), an open source framework written in Java [26]. Figure 3 shows a comparison of the accuracy of the NBCS, HT, and HOT algorithms. As demonstrated in Figure 3, the NBCS algorithm has a stable accuracy greater than 95%. The accuracy of the HOT algorithm is very close to that of the HT algorithm. When there are a relatively small number of samples, the HOT and HT algorithms each have an accuracy of only approximately 92%. As the number of learning samples increases, the accuracy of the HOT algorithm becomes slightly higher than that of the HT algorithm but is still lower than that of the NBCS algorithm by nearly 1%.

In addition to accuracy, three other statistical measures, namely, precision, recall, and specificity, were selected to

TABLE 1: Comparison of the NBCS, HT, and HOT algorithms for each index.

Algorithm	Precision	Recall	Specificity	Run time (ns)
NBCS	89.55%	93.28%	96.06%	350,846,550
HT	90.14%	86.02%	96.60%	1,682,211,860
HOT	90.04%	86.56%	96.69%	2,825,587,265

evaluate Android malware detection results. These three statistical measures are defined as follows:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP}, \\ \text{Recall} &= \frac{TP}{TP + FN}, \\ \text{Specificity} &= \frac{TN}{TN + FP}, \end{aligned} \quad (11)$$

where TP (true positive) represents the number of malware samples that are correctly detected; TN (true negative) represents the number of normal software samples that are correctly classified; FP (false positive) represents the number of normal samples that are detected as malware; and FN (false negative) represents the number of malware samples that are detected as normal software. As demonstrated in (11), precision represents the frequency of actual malware samples within the samples that are classified as malware; recall represents the frequency of samples that are correctly detected as malware within all the malware samples that are to be tested; and specificity represents the frequency of samples that are correctly classified as normal applications within all the normal application samples that are to be tested.

Table 1 shows the difference in each index between the NBCS algorithm and the comparative algorithms. As demonstrated in Table 1, the accuracy of each of the HT and HOT algorithms is higher than that of the NBCS algorithm by 0.50-0.6%, whereas the recall of the NBCS algorithm is higher than that of each of the HT and HOT algorithms by nearly 7%. Thus, the NBCS algorithm has a significantly higher capability to detect malware than the HT and HOT algorithms. The specificity of the NBCS algorithm is slightly lower than that of the HT algorithm, but only by 0.54%. In terms of the run time, the NBCS algorithm is significantly superior to the two decision-tree-based algorithms. The run time of the NBCS algorithm is only one-fifth that of the HT algorithm. While the classification performance of the HOT algorithm is slightly better than that of the HT algorithm, the run time of the HOT algorithm is much longer than that of the HT algorithm.

4.4. Analysis of the Performance of the ENBCS Algorithm.

In the ensemble learning model, a feature subset of size m will be randomly selected for each subclassifier, and each subclassifier will sufficiently test the features, select features, and ultimately classify the samples within the feature subset selected. Figure 4 shows the accuracy of the entire ensemble learning model corresponding to various feature subset sizes. As demonstrated in Figure 4, if m is set to a relatively

FIGURE 4: Accuracy of the ENBCS algorithm corresponding to various feature subset sizes when no subclassifier elimination strategy is implemented.

small value, the accuracy of the ensemble learning model is relatively low. Analyzing each subclassifier in the ensemble learning model revealed that because the number of feature subsets that can be selected is relatively small, the number of effective features that each subclassifier can use after the feature selection process is too small, and consequently each classifier cannot effectively classify the samples. The ENBCS algorithm is based on the NBCS algorithm and classifies samples using the maximum voting method. If there are a relatively large number of subclassifiers with relatively low accuracy, the accuracy of the entire ensemble classifier will also be relatively low.

To prevent subclassifiers with relatively low accuracies from continuously affecting the ensemble learning model, a window-based subclassifier elimination strategy is proposed for the ENBCS algorithm to retrain subclassifiers with relatively poor classification performance. Figure 5 shows the accuracy of the ensemble learning model corresponding to various feature subset sizes after the subclassifier elimination strategy is implemented. When m is set to a relatively large value, the accuracy of the ensemble learning model can become stable more quickly, and the impact of the value of m on the accuracy of the ensemble learning model is also relatively insignificant. When m is set to a relatively small value, the accuracy of the ensemble learning model increases significantly after the elimination strategy is implemented; in addition, as the number of samples continues to increase, the subclassifier elimination strategy can help continuously increase the accuracy of the ensemble learning model when m remains the same. In subsequent experiments, m is set to 150.

The ENBCS algorithm is also compared with other streaming data-based ensemble classifier algorithms, namely,

FIGURE 5: Accuracy of the ENBCS algorithm corresponding to various feature subset sizes.

FIGURE 6: Comparison of the accuracy of the ENBCS algorithm and the comparative algorithms.

the Online Bagging algorithm [27], the online boosting algorithm [27], the online coordinate boosting algorithm [28], the adaptive-size HT bagging (BagASHT) algorithm [29], the Accuracy-Weighted Ensemble algorithm [16], and the Accuracy Updated Ensemble algorithm [30]. These ensemble classifier algorithms are realized in MOA. Figure 6 shows the accuracy of each ensemble classifier algorithm. As demonstrated in Figure 6, the Online Bagging algorithm, the online boosting algorithm, the online coordinate

FIGURE 7: Impact of the subclassifier elimination strategy on the accuracy of the ENBCS for each data block when concept drift occurs in streaming data.

boosting algorithm, and the BagASHT algorithms have similar accuracies (in the range of 92%–94%). When there are a relatively small number of samples, the Accuracy-Weighted Ensemble algorithm and the Accuracy Updated Ensemble algorithm each have an accuracy of only 70%. As the number of learning samples increases, the accuracy of the Accuracy Updated Ensemble algorithm exceeds 91%, whereas the accuracy of the Accuracy-Weighted Ensemble algorithm remains below 90%. In comparison, the accuracy of the proposed ENBCS algorithm exceeds 96%, which is significantly higher than that of each of the comparative algorithms.

4.5. Concept Drift Detection. The previous experiments were conducted to examine the performance of the NBCS algorithm and the ensemble learning model under stable data source distribution conditions. The subclassifier elimination strategy implemented in the ENBCS algorithm has another important function: solving the concept drift problem. When concept drift occurs, because of the change in the data source distribution, subclassifier models that were trained on previous samples will fail to effectively examine new test samples. The concept drift problem can also be solved by examining the accuracy of the subclassifiers within the window in real time and retraining subclassifiers using the data within the window. Figure 7 shows the impact of the subclassifier elimination strategy on the accuracy of the ensemble learning model when concept drift occurs in streaming data. In this experiment, the NBCS algorithm window size is set to 200, the data block size (M) is set to 100, and the detection window size (K) is set to 5. As demonstrated in Figure 7, the accuracy of the ensemble learning model decreases suddenly when classifying test samples in the 29th data block. This is because the algorithm window size is set to 200 and

only the last 2,800 samples in the first part of the data are tested; consequently, the 29th data block is the data block in which concept drift begins to occur. By using the subclassifier elimination strategy, the concept drift is detected after the ensemble learning model completes the classification of the samples in the 31st data block, and subclassifiers are retrained using the data within the window. As a result, the accuracy of the ensemble learning model increases again to a relatively high level. In comparison, the ensemble classifier without the subclassifier elimination strategy is unable to make correct adjustments.

In this section, several ensemble learning based algorithms for solving the concept drift problem, namely, the Online Bagging ADWIN algorithm, the LimAttClassifier algorithm [17], the LeveragingBag algorithm [31], the Accuracy-Weighted Ensemble algorithm, and the Accuracy Updated Ensemble algorithm, are selected for comparison. These algorithms are all realized in MOA. Figure 8 shows the accuracy of the ENBCS algorithm and the comparative algorithms. As demonstrated in Figure 8, before concept drift occurs, the accuracy of the ENBCS algorithm is higher than that of each of the comparative algorithms for most data blocks. The ENBCS algorithm has a mean accuracy of 96.29%. In comparison, the Online Bagging ADWIN algorithm and the LimAttClassifier algorithm have mean accuracies of 92.9% and 93.93%, respectively. Compared with the Online Bagging ADWIN algorithm and the LimAttClassifier algorithm, the LeveragingBag algorithm has a slightly higher mean accuracy at 95.79%. The Accuracy-Weighted Ensemble algorithm and the Accuracy Updated Ensemble algorithm have the lowest mean accuracies (89.57% and 91.79%, resp.). After detecting the concept drift, the ensemble learning model makes adjustments. The accuracy of the ENBCS algorithm reaches 86.80%, which is still higher than that of each of the comparative algorithms. The LimAttClassifier algorithm has an accuracy of 85.22%. The LeveragingBag algorithm has a slightly higher accuracy, at 86.61%. The Accuracy Updated Ensemble algorithm has an accuracy of 80.6%. The Accuracy-Weighted Ensemble algorithm and the Online Bagging ADWIN algorithm have the lowest accuracies (77.00% and 77.28%, resp.). Thus, in terms of accuracy, the ENBCS algorithm is superior to the comparative algorithms. However, in terms of concept drift detection, the detection point of the ENBCS algorithm occurs slightly later than that of each of the Online Bagging ADWIN algorithm, the LimAttClassifier algorithm, and the LeveragingBag algorithm. This is related to the statistic (T_i) selected for subclassifier elimination in the ENBCS algorithm. As demonstrated in (9), T_i is in inverse proportion to the variance of the accuracy for each data block within the window (S). When considerable noise occurs in a data block within the window, the accuracy for this data block will be relatively low, and the value of $|\bar{P}_i - \bar{P}_{en}|$ will increase. However, the value of S will also increase correspondingly, and there will be no significant change in the value of T_i . Thus, subclassifiers will not be retrained because of noise in the data. When concept drift occurs, the accuracy will be relatively low for several data blocks within the window. Subclassifiers will be retrained only when S has a relatively small value.

FIGURE 8: Accuracy of the ENBCS algorithm and the comparative algorithms for each data block before and after the occurrence of concept drift.

FIGURE 9: Frequency of use of each feature in malware and normal software before and after the occurrence of concept drift.

4.6. Analysis of Concept Drifts in Android Malware Detection. As demonstrated in Figure 7, if the subclassifier elimination strategy is not implemented in the ENBCS algorithm, when concept drift occurs, the accuracy of the ENBCS algorithm will decrease greatly. Table 2 shows a comparison for each index of the ENBCS algorithm before and after the occurrence of concept drift. As demonstrated in Table 2, when concept drift occurs, the ENBCS algorithm can still

effectively detect malware samples; however, a large number of normal software samples are also detected as malware. To understand this phenomenon, the features selected by the NBSC algorithm (shown in Table 3) are analyzed. Figure 9 shows the frequency of use of each feature in malware and normal software before and after the occurrence of concept drift. As demonstrated in Figure 9(a), before concept drift occurs, the features of each dimension used by the NBSC

TABLE 2: Comparison of each index of the ENBCS algorithm before and after the occurrence of concept drift.

	Precision	Recall	Specificity	Accuracy
Before the occurrence of concept drift	94.44%	91.40%	98.05%	96.29%
After the occurrence of concept drift	28.28%	91.45%	16.03%	36.13%

TABLE 3: Features selected by the NBCS algorithm before and after the occurrence of concept drift.

Permission	Action	Code
ACCESS_WIFI_STATE	android.intent.action.VIEW	SmsManager
INSTALL_PACKAGES	android.intent.action.DIAL	getSimSerialNumber
INSTALL_SHORTCUT	android.intent.action.CALL	chmod
READ_HISTORY_BOOKMARKS	android.intent.action.SENDTO	abortBroadcast
READ_PHONE_STATE	android.intent.action.SEND	getLineNumber
READ_SMS	android.intent.action.WEB_SEARCH	/system/bin
RECEIVE_SMS	android.intent.action.SCREEN_OFF	getSimOperator
SEND_SMS	android.intent.action.SCREEN_ON	mount
UNINSTALL_SHORTCUT	android.intent.action.USER_PRESENT	KeySpec
WRITE_HISTORY_BOOKMARKS	android.intent.action.BOOT_COMPLETED	getNetworkOperator
	android.intent.action.PACKAGE_ADDED	SecretKey
	android.intent.action.BATTERY_CHANGED	

algorithm are used at significantly higher frequencies in malware than in normal software. According to (2), if a data sample uses a relatively large number of features of this type, this data sample will be identified by the NBCS algorithm as malware. As demonstrated in Figure 9(b), these features are used in normal software at considerably higher frequencies after concept drift occurs. Figure 9(a) shows the data distribution learned by the classifier. When the data distribution changes to the one shown in Figure 9(b), a large number of normal software programs will be classified as malware. This leads to the results shown in Figure 7 and Table 2. Therefore, in a streaming data environment, a concept drift detection method needs to be implemented to update the learning model in a timely fashion.

5. Summary and Outlook

In this work, the concept drift phenomenon in Android malware detection is analyzed. A classifier capable of adapting to streaming data, NBCS, is proposed. NBCS is formed by incorporating detection technology into a Naive Bayes classifier. This ensures that the training samples obtained are sufficient to represent the entire sample space, and NBCS can be used to detect Android malware. Android malware exhibits certain identifiable feature utilization tendencies. On this basis, a feature selection algorithm is added to NBCS. NBCS is also compared with the HT and HOT algorithms. The experimental results show that NBCS exhibits higher accuracy and higher operating efficiency compared with the other two algorithms. Based on NBCS, an ensemble learning model, ENBCS, is also proposed to solve the concept drift problem. ENBCS provides a random feature set to each subclassifier. By using a subclassifier elimination strategy, ENBCS monitors the accuracy of subclassifiers within a window in real time, eliminates the subclassifiers with relatively

poor performance, and subsequently retrains subclassifiers using the data within the window. The experimental results show that ENBCS can effectively detect Android malware, and its accuracy is higher than that of comparative algorithms.

While good results are achieved by using ENBCS to detect Android malware and solve the concept drift problem, improvements can be made in the following areas. First, the features used in this work originate from the permissions, actions, and codes used in Android applications. The features of the code layer are mainly bytecode written in Java. However, a large number of applications have now started to use the Java Native Interface (JNI) to call dynamic link libraries written in C++ to achieve higher performance and higher-level system calls. Features in this part of the code layer are difficult to procure. Second, the subclassifier elimination strategy implemented in the ENBCS algorithm employs hypothesis testing based on a sliding window with a fixed size. In future work, a new mechanism will be introduced to dynamically adjust the size of the sliding window. In addition, considering the rapid development of deep learning technology and its potential application in Android malware detection [32, 33], how to use deep learning based method to solve the concept drift problem in Android malware detection is also one of our future works.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under the Grants nos. 61379151U1636219 and U1636101 and the Natural Science

Research Project of Colleges and Universities in Anhui Province under the Grant no. KJ2017A734.

References

- [1] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine Learning*, vol. 1, no. 3, pp. 317–354, 1986.
- [2] X. Zhang, D. Hu, Y. Fan, and K. Yu, "A novel android malware detection method based on markov blanket," in *Proceedings of the 1st IEEE International Conference on Data Science in Cyberspace, DSC 2016*, pp. 347–352, June 2016.
- [3] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pp. 95–109, San Francisco, Calif, USA, May 2012.
- [4] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 97–106, San Francisco, Calif, USA, August 2001.
- [5] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pp. 71–80, August 2000.
- [6] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski, "Decision trees for mining data streams based on the mcdiarmid's bound," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1272–1279, 2013.
- [7] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "Decision trees for mining data streams based on the gaussian approximation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 108–119, 2014.
- [8] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence—SBIA 2004: Proceedings of the 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29–October 1, 2004*, vol. 3171 of *Lecture Notes in Computer Science*, pp. 286–295, Springer, Berlin, Germany, 2004.
- [9] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method in," in *4th International Workshop on Knowledge Discovery from Data Streams (IWKDDs 2006)*, 2006.
- [10] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 7th SIAM International Conference on Data Mining*, pp. 443–448, April 2007.
- [11] I. Koychev, "Gradual forgetting for adaptation to concept drift," in *Proceedings of the ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning*, pp. 101–106, 2004.
- [12] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281–300, 2004.
- [13] P. Zhang, X. Zhu, and Y. Shi, "Categorizing and mining concept drifting data streams," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2008*, pp. 812–820, usa, August 2008.
- [14] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 377–382, New York, NY, USA, August 2001.
- [15] J. Kolter and M. Maloof, "Dynamic weighted majority: a new ensemble method for tracking concept drift," in *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM '03)*, pp. 123–130, Melbourne, Australia, November 2003.
- [16] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pp. 226–235, Washington, DC, USA, August 2003.
- [17] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer, "Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking," *Journal of Machine Learning Research*, vol. 13, pp. 225–240, 2010.
- [18] L. Yu and H. Liu, "Feature selection for high-dimensional data: a fast correlation-based filter solution," in *ICML*, vol. 3, pp. 856–863, Feature selection for high-dimensional data, A fast correlation-based filter solution, 2003.
- [19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in FORTRAN*, Cambridge University Press, 1992.
- [20] D. Opitz and R. Maclin, "Popular ensemble methods: an empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.
- [21] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21–45, 2006.
- [22] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 1–39, 2010.
- [23] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 281–294, June 2012.
- [24] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC 2012*, pp. 1457–1462, ita, March 2012.
- [25] B. Pfahringer, G. Holmes, and R. Kirkby, *Australasian Joint Conference on Artificial Intelligence*, Springer, 2007.
- [26] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.
- [27] N. C. Oza, "Online bagging and boosting," in *Proceedings of the 2005 IEEE international conference on Systems, man and cybernetics*, vol. 3, pp. 2340–2345, 2005.
- [28] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin, "Online coordinate boosting," in *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009*, pp. 1354–1361, jpn, October 2009.
- [29] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New ensemble methods for evolving data streams," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*, pp. 139–147, Paris, France, July 2009.
- [30] D. Brzeziński and J. Stefanowski, "Accuracy updated ensemble for data streams with concept drift," in *Hybrid Artificial Intelligent System: 6th International Conference, HAIS 2011, Wroclaw, Poland, May 23–25, 2011, Proceedings, Part II*, E. Corchado, M. Kurzynski, and M. Wozniak, Eds., vol. 6679 of *Lecture Notes in Computer Science*, pp. 155–163, Springer, Berlin, Germany, 2011.
- [31] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," *Lecture Notes in Computer Science*

(including subseries *Lecture Notes in Artificial Intelligence* and *Lecture Notes in Bioinformatics*), vol. 6321, no. 1, pp. 135–150, 2010.

- [32] X. Su, D. Zhang, W. Li, and K. Zhao, “A deep learning approach to android malware feature learning and detection,” in *Proceedings of the Joint 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 10th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Symposium on Parallel and Distributed Processing with Applications, IEEE TrustCom/BigDataSE/ISPA 2016*, pp. 244–251, chn, August 2016.
- [33] R. Nix and J. Zhang, “Classification of Android apps and malware using deep neural networks,” in *Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1871–1878, Anchorage, AK, USA, May 2017.